GStreamer on Android

Who are we?

A short Introduction to GStreamer

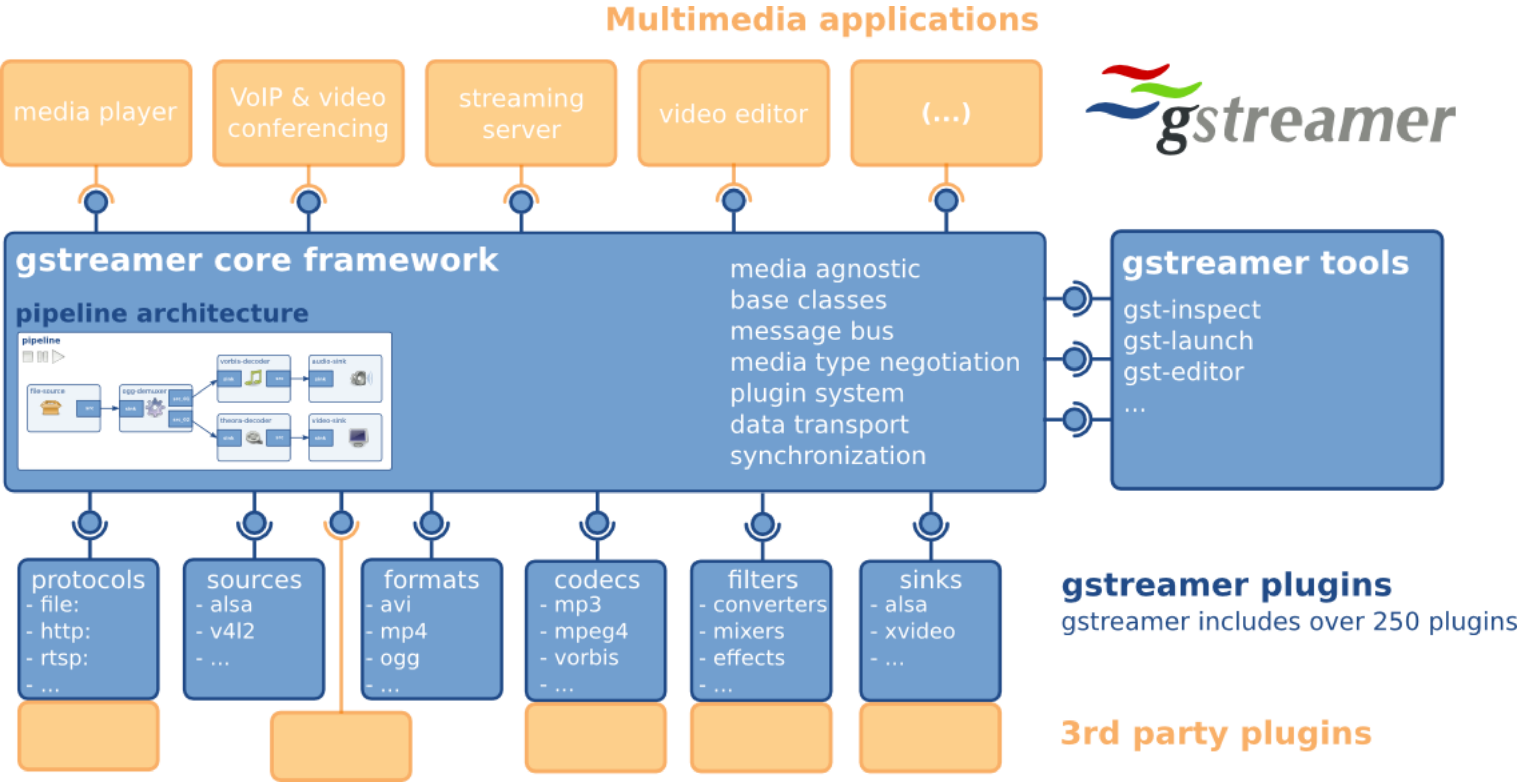
Just a short overview for those who don't know it yet

- Pipeline based multimedia framework
- Cross platform, open source
- Bindings for many languages
- Stable API/ABI

LGPL
Runs on Linux, Solaris, *BSD, OSX, Windows, …
x86, PPC, ARM, SPARC, …
Python, C++, .NET, Perl, Ruby, …
0.10 stable since >5 years, new 1.0 series

- Flexible and extensible design
- Plugin-based architecture
- Easy to integrate with other software
- Active developer and user community

Pipeline, elements
Media-agnostic core, media-aware plugins and libraries
All media-specific code in plugins
Only include what you need, add things later
Addition of 3rd party plugins, commercial/proprietary things
Simplifies license and patent nightmares
Integration into apps/libs, integration of libs (ffmpeg) into GStreamer



Multimedia applications

media player

VoIP & video conferencing

streaming server

video editor

(...)

gstreamer

gstreamer core framework

pipeline architecture

media agnostic
base classes
message bus
media type negotiation
plugin system
data transport
synchronization

gstreamer tools

gst-inspect
gst-launch
gst-editor
...

protocols
- file:
- http:
- rtsp:
- ...

sources
- alsa
- v4l2
- ...

formats
- avi
- mp4
- ogg
- ...

codecs
- mp3
- mpeg4
- vorbis
- ...

filters
- converters
- mixers
- effects
- ...

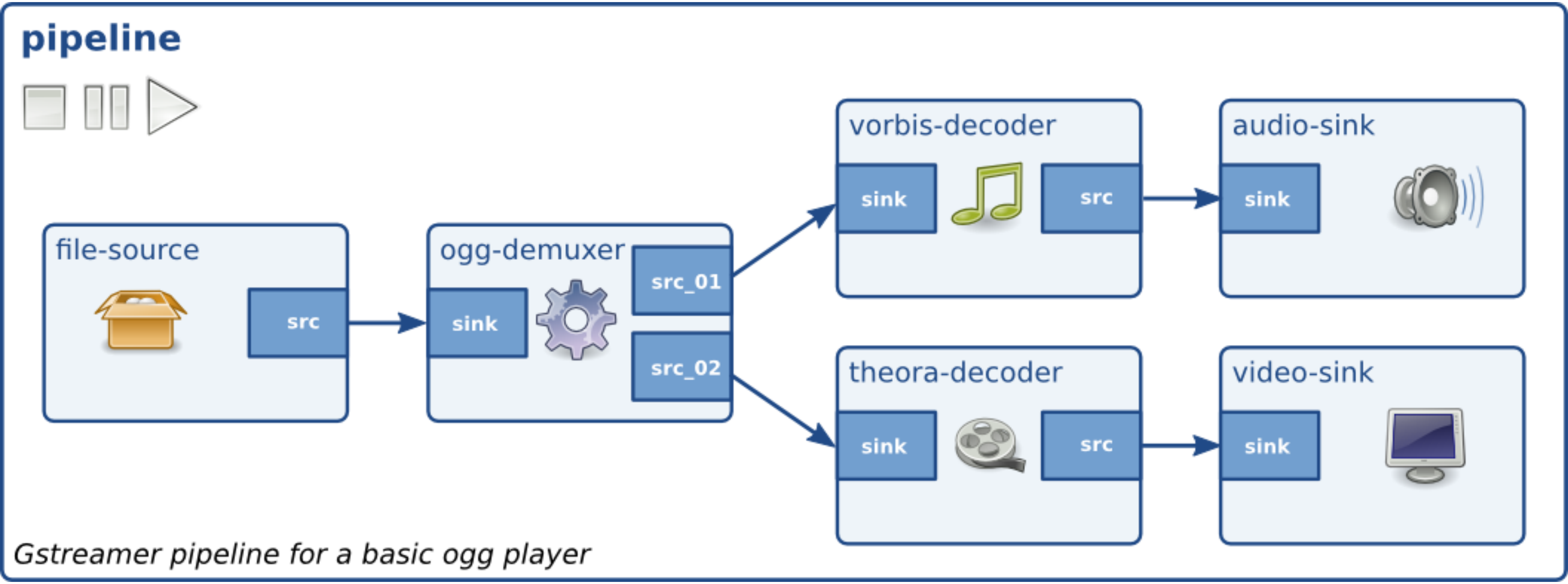
sinks
- alsa
- xvideo
- ...

gstreamer plugins

gstreamer includes over 250 plugins

3rd party plugins

Generic format negotiation mechanisms
Synchronization and data transport of media
Flexible communication between app and framework



Gstreamer pipeline for a basic ogg player

Very simple Ogg audio/video player pipeline

- Plugins for all important codecs and containers
- Proprietary plugins for patented codecs
- Plugins for different filters
- Hardware support
- Support for many different use cases

and also weird codecs/containers
Fluendo, Entropy Wave, device-specific plugins
Converters, mixers, effects, …
OpenMAX, OpenGL, V4L, VDPAU, VAAPI, …
Playback, encoding, realtime communication
Audio/video editing, signal processing, streaming server/client
Web browsers

- Used in many different applications on desktop platforms
- Used on many different devices by different companies

Linux (GNOME), OSX, Windows
Smartphones, tablets, video cameras, settop boxes, TVs,
video conference solutions, Android

GStreamer SDK

- Distribution of GStreamer with dependencies
- Available for Windows, OS X, Linux, Android
- IDE integration
- Starter documentation and tutorials
- Commercial support

fd.o project
Installers, packages
iOS planned later

Why use GStreamer on Android?

Android Multimedia stack

- Good multimedia API for playback and capture.
- Support for most common audio and video formats
- And a few streaming protocols

But...

Video players and camera capturers
Audio: AAC, MP3, FLAC, Vorbis, AMR, Midi, PCM
Video: H264, H264, AVC, VP8
Muxers: WebM, Matroska, MP4, OGG, Mpeg-TS
RTSP, HTTP, HLS

- We want much more than just playback or capture
- Some codecs and formats are not supported: ASF, DTS, or new codecs like Opus
- Other are device specific: WMA and WMV
- Only a few streaming protocols are supported: DASH, Smooth Streaming, RTP ?

We want to write any kind of application from Non-linear
to video editors, transcoders or media servers.
Opus, WMV and WMA and many other weird codecs
Only available on Tegra 3 devices
Smooth Streaming or DASH with GStreamer.

GStreamer has almost everything we need:

- Supports a very large number of formats.
- Support for more uses cases
- Multimedia backend re-usable across platforms.

Provide decoders, demuxer, encoders and demuxers
for a wide range for formats
A single multimedia for all platforms

Problems with using GStreamer on Android

- Plugin-based architecture -> too many shared libraries
- Android's dynamic linker limits the number of shared libraries per process.
- We have more than 262 shared libraries 😟
- Hard to easily distribute it in the Market
- Legal constraints with the LGPL and static linking.
- The NDK is limited: C library (BIONIC) and other libraries like OpenSL.

GStreamer itself only depends on glib, libxml2, libffi and libz,
but plugins pull-in many dependencies
android's dynamic linker has a hard-coded limit on the number
of .so files (shared libraries and/or plugins) you can load in a single process.
Android's linker is limited to 64, 96 and 128 shared libraries
Including all plugins we have 262 shared libraries
The LGPL requires a re-linking mechanism for statically linked
libraries.
Other API's are not even available in C like the
MediaCodec API

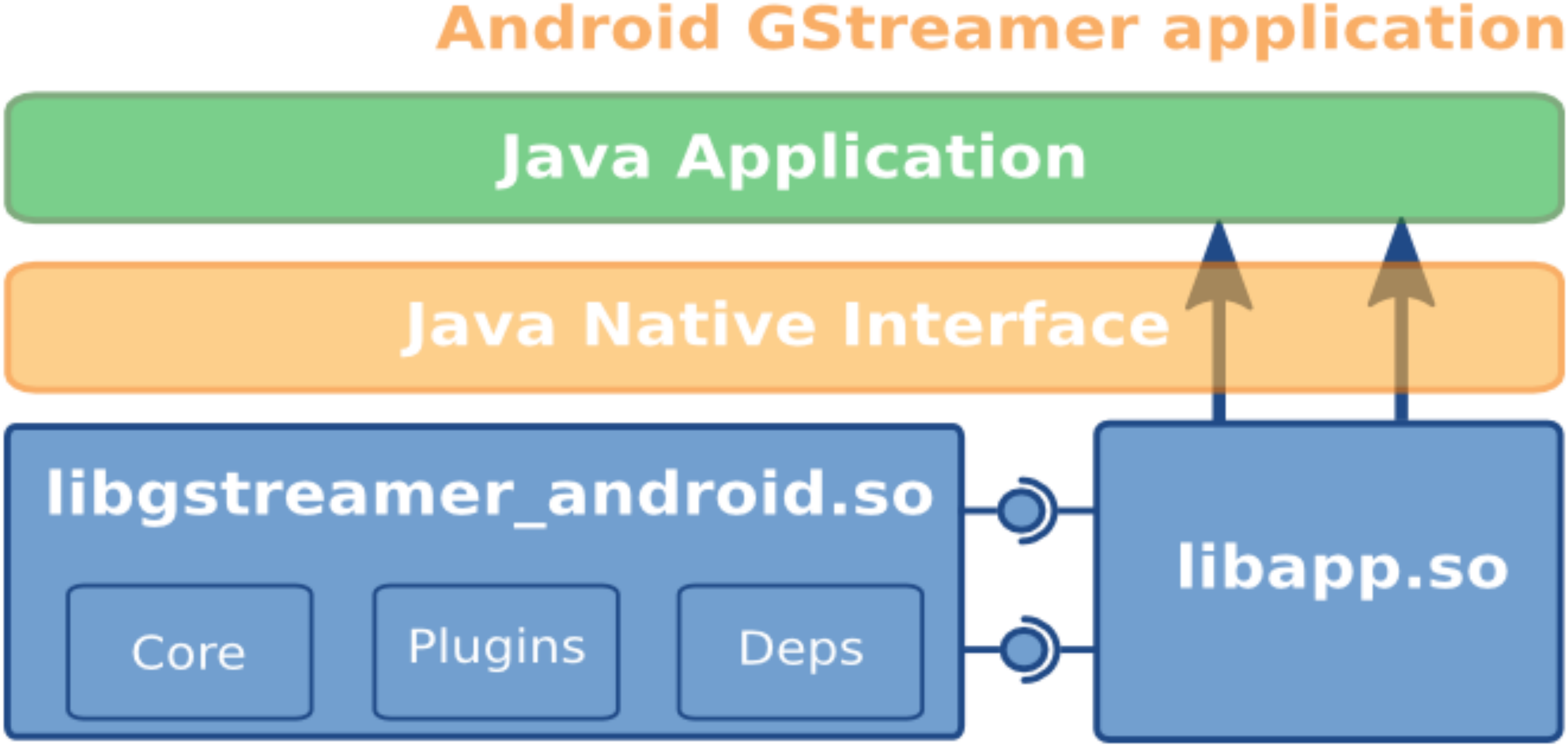
How we solved it..

- Static linking with re-locatable archives.
- A single shared library with everything: libgstreamer_android.so
- Integration with ndk-build to link this shared library:
  - Complies with the LGPL requirement.
  - Allows selecting only the plugins being used.

Static libraries built with -DPIC and -fPIC

Static linking inly includes oject libraries that are actually.

Easy to re-distribute and load in applications


Android GStreamer application
Java Application
Java Native Interface
libgstreamer_android.so
Core
Plugins
Deps
libapp.so

Building the SDK

- We use a build system called Cerbero.

- Same build system used to build the SDK in all platforms.

- Re-use of upstream packaging system.

- Native packaging:
  Windows .msi, OS X .pkg, RPM and DEB

- Easy to maintain

- Easy to add new packages or 3rd party plugins

We use cerbero, a build system developped for building the GStreamer SDK
Supports many platforms, cross-compilation and toolchain configuration.
Makefile, autotools and CMake. This saved us a lot ot time
compared to the old approach of porting the build to ndk-build

```
$ git clone git://anongit.freedesktop.org/gstreamer-sdk/cerbero

$ cerbero -c config/cross-android.cbc bootstrap

$ cerbero -c config/cross-android.cbc package gstreamer-sdk
```

Shows how simple and fast is rebuilding the sdk

Static plugins and modules

- GStreamer plugins and GIO modules must be handled in a different way.
- We are trying to get these changes upstream
- Static plugins need to be registered manually.

A bug open for glib, missing documentation for GStreamer
Instead of being loaded manually from path we must explicetly register them.

Integration with ndk-build

A set of makefiles that extend nkd-build's core to generate libgstreamer_android.so and link it to the application

From the point of view of application developers we tried to make things as easy as possible.

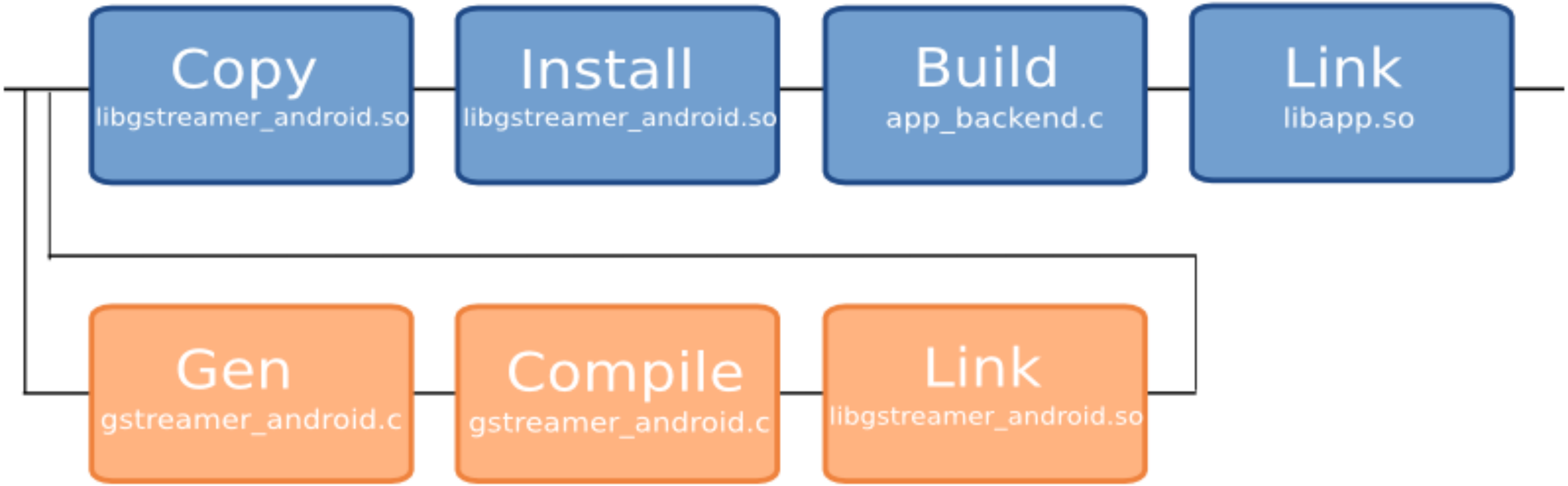
Example of Android.mk from the Android NDK samples

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := hello-jni
LOCAL_SRC_FILES := hello-jni.c
include $(BUILD_SHARED_LIBRARY)

include $(CLEAR_VARS)
```

jni/Android.mk modified to include GStreamer

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := hello-jni
LOCAL_SRC_FILES := hello-jni.c
LOCAL_SHARED_LIBRARIES := gstreamer_android
include $(BUILD_SHARED_LIBRARY)

include $(CLEAR_VARS)

include $(GSTREAMER_NDK_BUILD_PATH)/plugins.mk

GSTREAMER_SDK_ROOT := /home/cerbero/android_arm
GSTREAMER_PLUGINS =  $(GSTREAMER_PLUGINS_CORE)
                     $(GSTREAMER_PLUGINS_CODECS)
GSTREAMER_EXTRA_DEPS := json-glib-1.0

include $(GSTREAMER_NDK_BUILD_PATH)/gstreamer.mk
```

ndk-build build steps



Copy
libgstreamer_android.so

Install
libgstreamer_android.so

Build
app_backend.c

Link
libapp.so

Gen
gstreamer_android.c

Compile
gstreamer_android.c

Link
libgstreamer_android.so

gstreamer_android.c:
redirects GStreamer logs to logcat and adds an entry
point to initialize GStreamer and register static plugins
Libraries used from the C backend must be explicetly listed
to include the whole archive with --whole-archive, otherwise
the linker will not include the object files as no symbol
is used by the gstreamer plugins.)

- We use libtool libraries to resolve link deps

- Libtool can't be used for portability issues

- A small libtool replacement in makefiles + sed
  - Portable (works on Windows too)
  - Supports relocations of .la files
  - Much faster than libtool

Requires a unix-like shell on windows
equivalent to libtool --comand=link -static-libtool-libs
We can install the SDK everywhere as libtool is relocatable

And some stats...

- 171 plugins (same as for other platforms)
- Size of libstreamer_android.so
  - not stripped: 60 MB
  - stripped: 15 MB
  - not stripped without GStreamer debug: 55 MB
  - stripped without GStreamer debug: 13 MB

A library with everything is very cool, but at which cost?
Most of the time we don't need all plugins
This is using the tutorial plugins' list

New plugins

New plugins developed for the SDK on Android
To use only public API, add missing features

OpenGL ES / EGL Video Sink

- OpenGL ES/EGL only public, native API for video on Android
- Supports hardware accelerated colorspace conversion, scaling
- Usable on all Android devices
- Works like any other GStreamer video sink
- Allows embedding into Android applications
- Small and simple codebase

OpenGL not really made for video display but usable
Replacement for old surfaceflinger, non-public API
YUV, RGB, shaders
Also other platforms with OpenGL ES/EGL
Available since Android Gingerbread (2.3)

OpenSL ES Audio Sink/Source

- OpenSL ES only public native API for audio on Android
- Very limited implementation available on Android
- Usable on all Android devices
- Uses Android-specific API extensions
- Could support compressed formats later

Khronos standard (think: OpenGL, OpenMAX)

Replace old audioflinger based sink, non-public API

Very complex and powerful standard though, IMHO overengineered

- Mono, S16, 16kHz recording
- Mono/Stereo, U8/S16, up to 48kHz playback
- No device selection or any other more advanced features

Available since Android Gingerbread (2.3)

Minimal changes required to work on other platforms

android.media.MediaCodec Wrapper

- Be able to use device's codecs
- Uses Java API via JNI
- Java/JNI not performance problem
- Usable on all Android devices

Hardware acceleration, no worries about patent licenses

OpenMAX AL another option, very limited on Android

  only MPEGTS decoding

Very simple and powerful API

Very few method calls per frame

MediaCodec only small JNI wrapper around stagefright (C++)

Available since Android Jelly Bean (4.1)

- Implemented: audio/video decoders
- Encoders easy to add if necessary
- 1080p h264 easily possible, impossible in software
- Supported video codecs:
  h264/AVC, MPEG4, h263, MPEG2 and VP8
- Supported audio codecs:
  AAC, MP3, AMR-NB/WB, A-Law, µ-Law,
  Vorbis and FLAC

Hardware and software codecs
Tested so far on TI Ducati and NVidia Tegra3

Developing applications with the SDK

- GStreamer projects can be built using the regular tools

- For Eclipse: using the wizard and project→Android Tools→Add Native Support

- Command line: using the standard Ant build command

- jni/Android.mk must be updated for GStreamer

```
LOCAL_PATH := $(call my-dir)

include $(CLEAR_VARS)

LOCAL_MODULE    := hello-jni
LOCAL_SRC_FILES := hello-jni.c
LOCAL_SHARED_LIBRARIES := gstreamer_android
include $(BUILD_SHARED_LIBRARY)

include $(CLEAR_VARS)

include $(GSTREAMER_NDK_BUILD_PATH)/plugins.mk

GSTREAMER_SDK_ROOT := /home/cerbero/android_arm
GSTREAMER_PLUGINS =  $(GSTREAMER_PLUGINS_CORE)
                     $(GSTREAMER_PLUGINS_CODECS)

include $(GSTREAMER_NDK_BUILD_PATH)/gstreamer.mk
```

- No Java bindings yet
- Multimedia backend is written in C
- Bind the backend API to use it in the application through JNI

- Bind backend registering dynamic methods with RegisterNatives
- Declare this new methods as dynamic in the Java side

```
/* List of implemented native methods */
static JNINativeMethod native_methods[] = {
  { "nativeInit", "()V", (void *) gst_native_init},
  { "nativeFinalize", "()V", (void *) gst_native_finalize},
  { "nativePlay", "()V", (void *) gst_native_play},
  { "nativePause", "()V", (void *) gst_native_pause},
  { "nativeClassInit", "()Z", (void *) gst_native_class_init}
};

/* Library initializer */
jint JNI_OnLoad(JavaVM *vm, void *reserved) {
  JNIEnv *env = NULL;

  java_vm = vm;

  if ((*vm)->GetEnv(vm, (void**) &env, JNI_VERSION_1_4) != JNI_OK) {
    __android_log_print (ANDROID_LOG_ERROR, "tutorial-2", "Could not retrieve JNIEnv");
    return 0;
  }
  jclass klass = (*env)->FindClass (env, "com/gst_sdk_tutorials/tutorial_2/Tutorial2");
  (*env)->RegisterNatives (env, klass, native_methods, G_N_ELEMENTS(native_methods));

  pthread_key_create (&current_jni_env, detach_current_thread);

  return JNI_VERSION_1_4;
}
```

- Glib's main loop is run in a separate thread
- Use Thread-Local Storage (TLS) for storing the JNI env
- Load libgstreamer_android.so in the application

Help with threads that are not called form Java

- 5 tutorials specific for Android:
  - Linking against GStreamer
  - A running pipeline
  - Video
  - A basic media player
  - A complete media player

- 25 other tutorials for introducing developers into GStreamer development.

GStreamer SDK

http://www.gstreamer.com

Documentation and tutorials

http://www.docs.gstreamer.com

¿Questions?

Thanks!

FLUENDO
Influencing the Multimedia World

Collabora